

Docket No. AUS9-2000-0546-US1

**TRANSFERRING FOREIGN PROTOCOLS ACROSS A SYSTEM AREA
NETWORK**

5

BACKGROUND OF THE INVENTION**CROSS REFERENCES TO RELATED APPLICATIONS AND PATENTS**

Delm
The present invention is related to applications
entitled A System Area Network of End-to-End Context via
10 Reliable Datagram Domains, serial no. 09692351^(kh), attorney
docket no. AUS9-2000-0625-US1, filed 10/19/2000; Method
and Apparatus for Pausing a Send Queue without Causing
Sympathy Errors, serial no. 09692340^(kh), attorney docket
no. AUS9-2000-0626-US1, filed 10/19/2000; Method and
15 Apparatus to Perform Fabric Management, serial no.
09692344^(kh), attorney docket no. AUS9-2000-0627-US1, filed
10/19/2000; End Node Partitioning using LMC for a System
Area Network, serial no. 09692351^(kh), attorney docket no.
AUS9-2000-0628-US1, filed 10/19/2000; Method and
20 Apparatus for Dynamic Retention of System Area Network
Management Information in Non-Volatile Store, serial no.
09692365^(kh), attorney docket no. AUS9-2000-0629-US1, filed
10/19/2000; Method and Apparatus for Retaining Network
Security Settings Across Power Cycles, serial no.
25 09692337^(kh), attorney docket no. AUS9-2000-0630-US1, filed
10/19/2000; serial no. 09692348^(kh), attorney docket no.
AUS9-2000-0631-US1, filed 10/19/2000; Method and
Apparatus for Reliably Choosing a Master Network Manager
During Initialization of a Network Computing System,
30 serial no. 09692346^(kh), attorney docket no.
AUS9-2000-0632-US1, filed 10/19/2000; Method and
Apparatus for Ensuring Scalable Mastership During

US 2002/036126 A1

Docket No. AUS9-2000-0546-US1

Initialization of a System Area Network, serial no.

09692341, attorney docket no. AUS9-2000-0633-US1 filed

10/14/2000; and Method and Apparatus for Using a Service ID for the Equivalent of a Port ID in a Network Computing

5 System, serial no. 09692352, attorney docket no.

AUS9-2000-0634-US1 filed 10/14/2000, all of which are assigned to the same assignee, and incorporated herein by reference.

10 1. Technical Field:

The present invention relates generally to an improved data processing system, and in particular to a method and apparatus for handling PCI transactions over a network that implements the Infiniband architecture.

15

2. Description of Related Art:

In a System Area Network (SAN), the hardware provides a message passing mechanism which can be used for Input/Output devices (I/O) and interprocess communications between general computing nodes (IPC). Consumers access SAN message passing hardware by posting send/receive messages to send/receive work queues on a SAN channel adapter (CA). The send/receive work queues (WQ) are assigned to a consumer as a queue pair (QP). The

25 messages can be sent over five different defined transport types: Reliable Connected (RC), Reliable datagram (RD), Unreliable Connected (UC), Unreliable Datagram (UD), and Raw Datagram (RawD). In addition there is a set of manufacturer definable operation codes that

30 allow for different companies to define custom packets that still have the same routing header layouts.

00734999-1200700

Docket No. AUS9-2000-0546-US1

Consumers retrieve the results of the defined messages from a completion queue (CQ) through SAN send and receive work completions (WC). The manufacturer definable operations are not defined as to whether or not they use the same queuing structure as the defined packet types. Regardless of the packet type, the source channel adapter takes care of segmenting outbound messages and sending them to the destination. The destination channel adapter takes care of reassembling inbound messages and placing them in the memory space designated by the destination's consumer. Two channel adapter types are present, a host channel adapter (HCA) and a target channel adapter (TCA). The host channel adapter is used by general purpose computing nodes to access the SAN fabric. Consumers use SAN verbs to access host channel adapter functions. The software that interprets verbs and directly accesses the channel adapter is known as the channel interface (CI).

One disadvantage to these SAN fabric networks is that the I/O adapters and devices connected to the network must utilize this data packet communication protocol. However, there are many devices, such as, PCI I/O adapters, that do not use this protocol for communications, but that would be desirable to include into a SAN network. Therefore, there is a need for a method, system, and apparatus for incorporating PCI and PCI-X Input/Output Adapters (IOA) for use with the SAN fabric.

Docket No. AUS9-2000-0546-US1

SUMMARY OF THE INVENTION

5 The present invention provides a method, system, and apparatus for processing foreign protocol requests, such as PCI transactions, across a system area network (SAN) utilizing a data packet protocol while maintaining the other SAN traffic. In one embodiment, a HCA receives a
10 request for a load or store operation from a processor to an I/O adapter using a protocol which is foreign to the system area network, such as a PCI bus protocol. The HCA encapsulates the request into a data packet and places appropriate headers and trailers in the data packet to
15 ensure that the data packet is delivered across the SAN fabric to an appropriate TCA to which the requested I/O adapter is connected. The TCA receives the data packet, determines that it contains a foreign protocol request, and decodes the data packet to obtain the foreign
20 protocol request. The foreign protocol request is then transmitted to the appropriate I/O adapter. In the other direction, Direct Memory Access and interrupt traffic from the I/O adapter to the system is received by the TCA using a foreign protocol. The TCA encapsulates the
25 request into a data packet and places appropriate headers and trailers in the data packet to ensure that the data packet is delivered across the SAN fabric to the appropriate HCA. The HCA receives the data packet, determines that it contains a foreign protocol request,
30 and decodes the data packet to obtain the foreign protocol request, and converts the request to the appropriate host transaction.

Docket No. AUS9-2000-0546-US1

5

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts a diagram of a network computing system in accordance with a preferred embodiment of the present invention;

Figure 2 depicts a functional block diagram of a host processor node in accordance with a preferred embodiment of the present invention;

Figure 3 depicts a diagram of a host channel adapter in accordance with a preferred embodiment of the present invention;

Figure 4 depicts a diagram illustrating processing of work requests in accordance with a preferred embodiment of the present invention;

Figure 5 depicts an illustration of a data packet in accordance with a preferred embodiment of the present invention;

Figure 6 is a diagram illustrating a portion of a network computing system in accordance with a preferred embodiment of the present invention;

Figure 7 is a diagram illustrating messaging

occurring during establishment of a connection in accordance with a preferred embodiment of the present invention;

Figure 9 depicts a flowchart illustrating an exemplary method of performing a load operation from a processor to a PCI IOA in accordance with the present invention;

Figure 11 depicts a flowchart illustrating a direct memory access read operation from a PCI I/O adapter to system memory across an InfiniBand connection in accordance with a preferred embodiment of the present invention.

Docket No. AUS9-2000-0546-US1

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides a network computing system having end nodes, switches, routers, and links interconnecting these components. The end nodes segment the message into packets and transmit the packets over the links. The switches and routers interconnects the end nodes and route the packets to the appropriate end node. The end nodes reassemble the packets into a message at the destination. With reference now to the figures and in particular with reference to **Figure 1**, a diagram of a network computing system is illustrated in accordance with a preferred embodiment of the present invention. The network computing system represented in **Figure 1** takes the form of a system area network (SAN) **100** and is provided merely for illustrative purposes, and the embodiments of the present invention described below can be implemented on computer systems of numerous other types and configurations. For example, computer systems implementing the present invention can range from a small server with one processor and a few input/output (I/O) adapters to massively parallel supercomputer systems with hundreds or thousands of processors and thousands of I/O adapters. Furthermore, the present invention can be implemented in an infrastructure of remote computer systems connected by an internet or intranet.

SAN 100 is a high-bandwidth, low-latency network interconnecting nodes within the network computing system. A node is any component attached to one or more links of a network and forming the origin and/or

Docket No. AUS9-2000-0546-US1

destination of messages within the network. In the depicted example, SAN 100 includes nodes in the form of host processor node 102, host processor node 104, redundant array independent disk (RAID) subsystem node 106, I/O chassis node 108, and PCI I/O Chassis node 184. The nodes illustrated in **Figure 1** are for illustrative purposes only, as SAN 100 can connect any number and any type of independent processor nodes, I/O adapter nodes, and I/O device nodes. Any one of the nodes can function as an endnode, which is herein defined to be a device that originates or finally consumes messages or frames in SAN 100.

In one embodiment of the present invention, an error handling mechanism in distributed computer systems is present in which the error handling mechanism allows for reliable connection or reliable datagram communication between end nodes in network computing system, such as SAN 100.

A message, as used herein, is an application-defined unit of data exchange, which is a primitive unit of communication between cooperating processes. A packet is one unit of data encapsulated by a networking protocol headers and/or trailer. The headers generally provide control and routing information for directing the frame through SAN. The trailer generally contains control and cyclic redundancy check (CRC) data for ensuring packets are not delivered with corrupted contents.

SAN 100 contains the communications and management infrastructure supporting both I/O and interprocessor communications (IPC) within a network computing system. The SAN 100 shown in **Figure 1** includes a switched

Docket No. AUS9-2000-0546-US1

communications fabric **100**, which allows many devices to concurrently transfer data with high-bandwidth and low latency in a secure, remotely managed environment.

Endnodes can communicate over multiple ports and utilize
5 multiple paths through the SAN fabric. The multiple ports and paths through the SAN shown in **Figure 1** can be employed for fault tolerance and increased bandwidth data transfers.

The SAN **100** in **Figure 1** includes switch **112**, switch
10 **114**, switch **146**, and router **117**. A switch is a device that connects multiple links together and allows routing of packets from one link to another link within a subnet using a small header Destination Local Identifier (DLID) field. A router is a device that connects multiple
15 subnets together and is capable of routing frames from one link in a first subnet to another link in a second subnet using a large header Destination Globally Unique Identifier (DGUID).

In one embodiment, a link is a full duplex channel
20 between any two network fabric elements, such as endnodes, switches, or routers. Example suitable links include, but are not limited to, copper cables, optical cables, and printed circuit copper traces on backplanes and printed circuit boards.

25 For reliable service types, endnodes, such as host processor endnodes and I/O adapter endnodes, generate request packets and return acknowledgment packets. Switches and routers pass packets along, from the source to the destination. Except for the variant CRC trailer
30 field which is updated at each stage in the network, switches pass the packets along unmodified. Routers

Docket No. AUS9-2000-0546-US1

update the variant CRC trailer field and modify other fields in the header as the packet is routed.

In SAN 100 as illustrated in **Figure 1**, host processor node 102, host processor node 104, RAID I/O subsystem 106, I/O chassis 108, and PCI I/O Chassis 184 include at least one channel adapter (CA) to interface to SAN 100. In one embodiment, each channel adapter is an endpoint that implements the channel adapter interface in sufficient detail to source or sink packets transmitted on SAN fabric 100. Host processor node 102 contains channel adapters in the form of host channel adapter 118 and host channel adapter 120. Host processor node 104 contains host channel adapter 122 and host channel adapter 124. Host processor node 102 also includes central processing units 126-130 and a memory 132 interconnected by bus system 134. Host processor node 104 similarly includes central processing units 136-140 and a memory 142 interconnected by a bus system 144.

Host channel adapter 118 provides a connection to switch 112, host channel adapters 120 and 122 provide a connection to switches 112 and 114, and host channel adapter 124 provides a connection to switch 114.

In one embodiment, a host channel adapter is implemented in hardware. In this implementation, the host channel adapter hardware offloads much of central processing unit and I/O adapter communication overhead. This hardware implementation of the host channel adapter also permits multiple concurrent communications over a switched network without the traditional overhead associated with communicating protocols. In one

Docket No. AUS9-2000-0546-US1

embodiment, the host channel adapters and SAN 100 in **Figure 1** provide the I/O and interprocessor communications (IPC) consumers of the network computing system with zero processor-copy data transfers without involving the operating system kernel process, and employs hardware to provide reliable, fault tolerant communications.

As indicated in **Figure 1**, router 117 is coupled to wide area network (WAN) and/or local area network (LAN) connections to other hosts or other routers.

The I/O chassis 108 in **Figure 1** includes a switch 146 and multiple I/O modules 148-156. In these examples, the I/O modules take the form of adapter cards. Example adapter cards illustrated in **Figure 1** include a SCSI adapter card for I/O module 148; an adapter card to fiber channel hub and fiber channel-arbitrated loop(FC-AL) devices for I/O module 152; an ethernet adapter card for I/O module 150; a graphics adapter card for I/O module 154; and a video adapter card for I/O module 156. Any known type of adapter card can be implemented. I/O adapters also include a switch in the I/O adapter backplane to couple the adapter cards to the SAN fabric. These modules contain target channel adapters 158-166.

In this example, RAID subsystem node 106 in **Figure 1** includes a processor 168, a memory 170, a target channel adapter (TCA) 172, and multiple redundant and/or striped storage disk unit 174. Target channel adapter 172 can be a fully functional host channel adapter.

PCI I/O Chassis node 184 includes a TCA 186 and multiple PCI Input/Output Adapters (IOA) 190-192

Docket No. AUS9-2000-0546-US1

connected to TCA 186 via PCI bus 188. In these examples, the IOAs take the form of adapter cards. Example adapter cards illustrated in Figure 1 include a modem adapter card 190 and serial adapter card 192. TCA 186 encapsulates PCI transaction requests or responses received from PCI IOAs 190-192 into data packets for transmission across the SAN fabric 100 to an HCA, such as HCA 118. HCA 118 determines whether received data packets contain PCI transmissions and, if so, decodes the data packet to retrieve the encapsulated PCI transaction request or response, such as a DMA write or read operation. HCA 118 sends it to the appropriate unit, such as memory 132. If the PCI transaction was a DMA read request, the HCA then receives the response from the memory, such as memory 132, encapsulates the PCI response into a data packet, and sends the data packet back to the requesting TCA 186 across the SAN fabric 100. the TCA then decodes the PCI transaction from the data packet and sends the PCI transaction to PCI IOA 190 or 192 across PCI bus 188.

Similarly, store and load requests from a processor, such as, for example, CPU 126, to a PCI IOA, such as PCI IOA 190 or 192 are encapsulated into a data packet by the HCA 118 for transmission to the TCA 186 corresponding to the appropriate PCI IOA 190 or 192 across SAN fabric 100. The TCA 186 decodes the data packet to retrieve the PCI transmission and transmits the PCI store or load request and data to PCI IOA 190 or 192 via PCI bus 188. If the request is a load request, the TCA 186 then receives a response from the PCI IOA 190 or 192 which the TCA

Docket No. AUS9-2000-0546-US1

encapsulates into a data packet and transmits over the SAN fabric **100** to HCA **118** which decodes the data packet to retrieve the PCI data and commands and sends the PCI data and commands to the requesting CPU **126**. Thus, PCI
5 adapters may be connected to the SAN fabric **100** of the present invention.

SAN **100** handles data communications for I/O and interprocessor communications. SAN **100** supports high-bandwidth and scalability required for I/O and also
10 supports the extremely low latency and low CPU overhead required for interprocessor communications. User clients can bypass the operating system kernel process and directly access network communication hardware, such as host channel adapters, which enable efficient message
15 passing protocols. SAN **100** is suited to current computing models and is a building block for new forms of I/O and computer cluster communication. Further, SAN **100** in **Figure 1** allows I/O adapter nodes to communicate among themselves or communicate with any or all of the
20 processor nodes in network computing system. With an I/O adapter attached to the SAN **100**, the resulting I/O adapter node has substantially the same communication capability as any host processor node in SAN **100**.

Turning next to **Figure 2**, a functional block diagram
25 of a host processor node is depicted in accordance with a preferred embodiment of the present invention. Host processor node **200** is an example of a host processor node, such as host processor node **102** in **Figure 1**.

In this example, host processor node **200** shown in **Figure**
30 **2** includes a set of consumers **202-208** and one or more

Docket No. AUS9-2000-0546-US1

PCI/PCI-X device drivers 230, which are processes executing on host processor node 200. Host processor node 200 also includes channel adapter 210 and channel adapter 212. Channel adapter 210 contains ports 214 and 216 while channel adapter 212 contains ports 218 and 220. Each port connects to a link. The ports can connect to one SAN subnet or multiple SAN subnets, such as SAN 100 in **Figure 1**. In these examples, the channel adapters take the form of host channel adapters.

10 Consumers 202-208 transfer messages to the SAN via the verbs interface 222 and message and data service 224. A verbs interface is essentially an abstract description of the functionality of a host channel adapter. An operating system may expose some or all of the verb
15 functionality through its programming interface. Basically, this interface defines the behavior of the host. Additionally, host processor node 200 includes a message and data service 224, which is a higher level interface than the verb layer and is used to process
20 messages and data received through channel adapter 210 and channel adapter 212. Message and data service 224 provides an interface to consumers 202-208 to process messages and other data. In addition, the channel adapter 210 and channel adapter 212 may receive load and
25 store instructions from the processors which are targeted for PCI IOAs attached to the SAN. These bypass the verb layer, as shown in **Figure 2**.

With reference now to **Figure 3**, a diagram of a host channel adapter is depicted in accordance with a
30 preferred embodiment of the present invention. Host

Docket No. AUS9-2000-0546-US1

channel adapter **300** shown in **Figure 3** includes a set of queue pairs (QPs) **302-310**, which are one means used to transfer messages to the host channel adapter ports **312-316**. Buffering of data to host channel adapter ports **312-316** is channeled through virtual lanes (VL) **318-334** where each VL has its own flow control. Subnet manager configures channel adapters with the local addresses for each physical port, i.e., the port's LID. Subnet manager agent (SMA) **336** is the entity that communicates with the subnet manager for the purpose of configuring the channel adapter. Memory translation and protection (MTP) **338** is a mechanism that translates virtual addresses to physical addresses and to validate access rights. Direct memory access (DMA) **340** provides for direct memory access operations using memory **340** with respect to queue pairs **302-310**.

A single channel adapter, such as the host channel adapter **300** shown in **Figure 3**, can support thousands of queue pairs. By contrast, a target channel adapter in an I/O adapter typically supports a much smaller number of queue pairs.

Each queue pair consists of a send work queue (SWQ) and a receive work queue. The send work queue is used to send channel and memory semantic messages. The receive work queue receives channel semantic messages. A consumer calls an operating-system specific programming interface, which is herein referred to as verbs, to place work requests (WRs) onto a work queue.

The method of using the SAN to send foreign protocols across the network, as defined herein, does not use the queue pairs, but instead bypasses these on the

Docket No. AUS9-2000-0546-US1

way to the SAN. These foreign protocols, do, however, use the virtual lanes (e.g., virtual lane 334). Many protocols require special ordering of operations in order to prevent deadlocks. A deadlock can occur, for example, when two operations have a dependency on one another for completion and neither can complete before the other completes. The PCI specification, for example, requires certain ordering be followed for deadlock avoidance. The virtual lane mechanism can be used when one operation needs to bypass another in order to avoid a deadlock. In this case, the different operations that need to bypass are assigned to different virtual lanes.

With reference now to **Figure 4**, a diagram illustrating processing of work requests is depicted in accordance with a preferred embodiment of the present invention. In **Figure 4**, a receive work queue 400, send work queue 402, and completion queue 404 are present for processing requests from and for consumer 406. These requests from consumer 402 are eventually sent to hardware 408. In this example, consumer 406 generates work requests 410 and 412 and receives work completion 414. As shown in **Figure 4**, work requests placed onto a work queue are referred to as work queue elements (WQEs).

Send work queue 402 contains work queue elements (WQEs) 422-428, describing data to be transmitted on the SAN fabric. Receive work queue 400 contains work queue elements (WQEs) 416-420, describing where to place incoming channel semantic data from the SAN fabric. A work queue element is processed by hardware 408 in the host channel adapter.

Docket No. AUS9-2000-0546-US1

The verbs also provide a mechanism for retrieving completed work from completion queue **404**. As shown in **Figure 4**, completion queue **404** contains completion queue elements (CQEs) **430-436**. Completion queue elements
 5 contain information about previously completed work queue elements. Completion queue **404** is used to create a single point of completion notification for multiple queue pairs. A completion queue element is a data structure on a completion queue. This element describes a completed
 10 work queue element. The completion queue element contains sufficient information to determine the queue pair and specific work queue element that completed. A completion queue context is a block of information that contains pointers to, length, and other information
 15 needed to manage the individual completion queues.

Example work requests supported for the send work queue **402** shown in **Figure 4** are as follows. A send work request is a channel semantic operation to push a set of local data segments to the data segments referenced by a
 20 remote node's receive work queue element. For example, work queue element **428** contains references to data segment 4 **438**, data segment 5 **440**, and data segment 6 **442**. Each of the send work request's data segments contains a virtually contiguous memory region. The
 25 virtual addresses used to reference the local data segments are in the address context of the process that created the local queue pair.

A remote direct memory access (RDMA) read work request provides a memory semantic operation to read a
 30 virtually contiguous memory space on a remote node. A memory space can either be a portion of a memory region

Docket No. AUS9-2000-0546-US1

or portion of a memory window. A memory region references a previously registered set of virtually contiguous memory addresses defined by a virtual address and length. A memory window references a set of
5 virtually contiguous memory addresses which have been bound to a previously registered region.

The RDMA Read work request reads a virtually contiguous memory space on a remote endnode and writes the data to a virtually contiguous local memory space.
10 Similar to the send work request, virtual addresses used by the RDMA Read work queue element to reference the local data segments are in the address context of the process that created the local queue pair. For example, work queue element **416** in receive work queue **400**
15 references data segment 1 **444**, data segment 2 **446**, and data segment **448**. The remote virtual addresses are in the address context of the process owning the remote queue pair targeted by the RDMA Read work queue element.

A RDMA Write work queue element provides a memory
20 semantic operation to write a virtually contiguous memory space on a remote node. The RDMA Write work queue element contains a scatter list of local virtually contiguous memory spaces and the virtual address of the remote memory space into which the local memory spaces
25 are written.

A RDMA FetchOp work queue element provides a memory semantic operation to perform an atomic operation on a remote word. The RDMA FetchOp work queue element is a combined RDMA Read, Modify, and RDMA Write operation.
30 The RDMA FetchOp work queue element can support several read-modify-write operations, such as Compare and Swap if

00731900-130700
00000000-00000000

Docket No. AUS9-2000-0546-US1

equal.

A bind (unbind) remote access key (R_Key) work queue element provides a command to the host channel adapter hardware to modify (destroy) a memory window by

- 5 associating (disassociating) the memory window to a memory region. The R_Key is part of each RDMA access and is used to validate that the remote process has permitted access to the buffer.

- In one embodiment, receive work queue 400 shown in
10 **Figure 4** only supports one type of work queue element, which is referred to as a receive work queue element. The receive work queue element provides a channel semantic operation describing a local memory space into which incoming send messages are written. The receive
15 work queue element includes a scatter list describing several virtually contiguous memory spaces. An incoming send message is written to these memory spaces. The virtual addresses are in the address context of the process that created the local queue pair.

- 20 For interprocessor communications, a user-mode software process transfers data through queue pairs directly from where the buffer resides in memory. In one embodiment, the transfer through the queue pairs bypasses the operating system and consumes few host instruction
25 cycles. Queue pairs permit zero processor-copy data transfer with no operating system kernel involvement. The zero processor-copy data transfer provides for efficient support of high-bandwidth and low-latency communication.

- 30 When a queue pair is created, the queue pair is set to provide a selected type of transport service. In one

Docket No. AUS9-2000-0546-US1

embodiment, a network computing system implementing the present invention supports four types of transport services.

Reliable and Unreliable connected services associate
5 a local queue pair with one and only one remote queue pair. Connected services require a process to create a queue pair for each process which is to communicate with over the SAN fabric. Thus, if each of N host processor nodes contain P processes, and all P processes on each
10 node wish to communicate with all the processes on all the other nodes, each host processor node requires $P^2 \times (N - 1)$ queue pairs. Moreover, a process can connect a queue pair to another queue pair on the same host channel adapter.

Reliable datagram service associates a local end-end
15 (EE) context with one and only one remote end-end context. The reliable datagram service permits a client process of one queue pair to communicate with any other queue pair on any other remote node. At a receive work
20 queue, the reliable datagram service permits incoming messages from any send work queue on any other remote node. The reliable datagram service greatly improves scalability because the reliable datagram service is connectionless. Therefore, an endnode with a fixed
25 number of queue pairs can communicate with far more processes and endnodes with a reliable datagram service than with a reliable connection transport service. For example, if each of N host processor nodes contain P processes, and all P processes on each node wish to
30 communicate with all the processes on all the other nodes, the reliable connection service requires $P^2 \times (N -$

Docket No. AUS9-2000-0546-US1

1) queue pairs on each node. By comparison, the connectionless reliable datagram service only requires P queue pairs + $(N - 1)$ EE contexts on each node for exactly the same communications.

5 The unreliable datagram service is connectionless. The unreliable datagram service is employed by management applications to discover and integrate new switches, routers, and endnodes into a given network computing system. The unreliable datagram service does not provide
10 the reliability guarantees of the reliable connection service and the reliable datagram service. The unreliable datagram service accordingly operates with less state information maintained at each endnode.

 The description of the present invention turns now
15 to identifying service during connection establishment. **Figures 5, 6, and 7** together illustrate how a service is identified during the connection establishment process.

 Turning next to **Figure 5**, an illustration of a data packet is depicted in accordance with a preferred
20 embodiment of the present invention. Message data **500** contains data segment 1 **502**, data segment 2 **504**, and data segment 3 **506**, which are similar to the data segments illustrated in **Figure 4**. In this example, these data segments form a packet **508**, which is placed into packet
25 payload **510** within data packet **512**. Additionally, data packet **512** contains CRC **514**, which is used for error checking. Additionally, routing header **516** and transport **518** are present in data packet **512**. Routing header **516** is used to identify source and destination ports for data
30 packet **512**. Transport header **518** in this example specifies the destination queue pair for data packet **512**.

007388-120700

Docket No. AUS9-2000-0546-US1

00734999-120700

Additionally, transport header **518** also provides information such as the operation code, packet sequence number, and partition for data packet **512**. The operating code identifies whether the packet is the first, last, intermediate, or only packet of a message. The operation code also specifies whether the operation is a send RDMA write, read, or atomic. The packet sequence number is initialized when communications is established and increments each time a queue pair creates a new packet.

10 Ports of an endnode may be configured to be members of one or more possibly overlapping sets called partitions.

In **Figure 6**, a diagram illustrating a portion of a network computing system is depicted in accordance with a preferred embodiment of the present invention. The network computing system **600** in **Figure 6** includes a host processor node **602**, a host processor node **604**, a SAN fabric **610**, and I/O which includes TCA **642** and IOA **646**. Host processor node **602** includes a host channel adapter (HCA) **606**. Host processor node **604** includes a host channel adapter (HCA) **608**. The network computing system in **Figure 6** includes a SAN fabric **610** which includes a switch **612** and a switch **614**. SAN fabric **610** in **Figure 6** includes a link coupling host channel adapter **606** to switch **612**; a link coupling switch **612** to switch **614**; a link coupling switch **612** to TCA **642**; and a link coupling host channel adapter **608** to switch **614**.

20

25

In the example transactions, host processor node **602** includes a client process A **616**. Host processor node **604** includes a client process B **618**. Client process A **616** interacts with host channel adapter hardware **606** through

30

Docket No. AUS9-2000-0546-US1

queue pair **620**. Client process B **618** interacts with host channel adapter **608** through queue pair **622**. Queue pair **620** and queue pair **622** are data structures. Queue pair **620** includes a send work queue **624** and a receive work queue **626**. Queue pair **622** includes a send work queue **628** and a receive work queue **630**. All of these queue pairs are unreliable datagram General Service Interface (GSI) queue pairs. GSI queue pairs are used for management including the connection establishment process.

10 Process A **616** initiates a connection establishment message request by posting send work queue elements to the send queue **624** of queue pair **620**. Such a work queue element is illustrated in **Figure 4** above. The message request of client process A **616** is referenced by a gather list contained in the send work queue element. Each of
15 the data segments in the gather list points to a virtually contiguous local memory region, which contains the Connection Management REQuest message. This message is used to request a connection between host channel
20 adapter **606** and host channel adapter **608**.

Each process residing in host node processor node **604** communicates to process B **618** the ServiceID which is associated to each specific process. Process B **618** will then compare the ServiceID of incoming REQ messages to
25 the ServiceID associated with each process.

Referring to **Figures 5** and **7**, like all other unreliable datagram (UD) messages, the REQ message sent by process A **616** is a single packet message. Host channel adapter **606** sends the REQ message contained in
30 the work queue element posted to queue pair **620**. The REQ

Docket No. AUS9-2000-0546-US1

message is destined for host channel adapter **608**, queue pair **622**, and contains a Service ID field. The ServiceID field is used by the destination to determine which consumer is associated to the Service ID. The REQ message is placed in the next available receive work queue element from the receive queue **630** of queue pair **622** in host channel adapter **608**. Process B **618** polls the completion queue and retrieves the completed receive work queue element from the receive queue **630** of queue pair **622** in host channel adapter **608**. The completed receive work queue element contains the REQ message process A **616** sent. Process B **618** then compares the ServiceID of the REQ message to the ServiceID of each process that has registered itself with process B **618**. Registering means that one software process identifies itself to another software process by some predetermined means, such as providing an identifying serviceID. If a match occurs, process B **618** passes the REQ message to the matching process. Otherwise the REQ message is rejected and process B **618** sends process A **616** a REJ message.

If the consumer accepts the messages, the consumer informs process B **618** that the connection establishment REQ has been accepted. Process B **618** responds to the connection establishment REQ by posting send work queue elements to the send queue **628** of queue pair **622**. Such a work queue element is illustrated in **Figure 4**. The message response of client process B **618** is referenced by a gather list contained in the send work queue element. Each of the data segments in the gather list point to a virtually contiguous local memory region, which contains

Docket No. AUS9-2000-0546-US1

the Connection Management REPlY message. This message is used to accept the connection establishment REQ. If the consumer rejects the message, the consumer informs process B **618** that the connection establishment REQ has
5 been rejected.

If either process B **618** or the consumer reject the message, process B **618** responds to the connection establishment REQ by posting send work queue elements to the send queue **628** of queue pair **622**. Such a work queue
10 element is illustrated in **Figure 4**. The message response of client process B **618** is referenced by a gather list contained in the send work queue element. Each of the data segments in the gather list point to a virtually contiguous local memory region, which contains the
15 Connection Management REJect message. This message is used to reject the connection establishment REQ.

Referring to **Figures 5** and **7**, the REP message sent by process B **618** is a single packet message. Host channel adapter **608** sends the REP message contained in the work
20 queue element posted to queue pair **622**. The REP message is destined for host channel adapter **606**, queue pair **620**, and contains acceptance of the previous REQ. The acceptance confirms that a process in host channel adapter **608** is indeed associated with the ServiceID sent
25 in the REQ message and lets process A **616** know which queue pair to use to communicate to the process associated with the ServiceID. Upon arriving successfully at host channel adapter **606**, the REQ message is placed in the next available receive work queue element from the
30 receive queue **626** of queue pair **620** in host channel

Docket No. AUS9-2000-0546-US1

adapter 606. Process A 616 polls the completion queue and retrieves the completed receive work queue element from the receive queue 626 of queue pair 620 in host channel adapter 606. The completed receive work queue element
5 contains the REP message sent by process B 618. Process A 616 now knows the service was valid, the queue pair to use to reach it, and that the connection was accepted. Process A 616 proceeds with the remainder of the connection establishment protocol.

10 Referring again to **Figure 6**, another example transaction is between PCI device driver 640 and PCI IOA 646. Store and load requests from the PCI device driver 640 to PCI IOA 646 are encapsulated into a data packet by the HCA 606 for transmission to the TCA 642 corresponding
15 to the PCI IOA 646 across SAN fabric 100. The TCA 642 decodes the data packet to retrieve the PCI transmission and transmits the PCI store or load request and data to PCI IOA 646 via PCI bus 644. If the request is a load request, the TCA 642 then receives a response from the
20 PCI IOA 646 which the TCA encapsulates into a data packet and transmits over the SAN fabric 100 to HCA 606 which decodes the data packet to retrieve the PCI data and commands and sends the PCI data and commands to the requesting PCI device driver 640.

25 In addition to data needing to be passed back and forth between the processing nodes and the I/O nodes, the I/O also has another mechanism that needs to pass between these endnodes, namely interrupts. Interrupts are used by the I/O adapters to signal their device driver
30 software that an operation is complete or that other

Docket No. AUS9-2000-0546-US1

servicing is required, for example error recovery. There are different protocols used for signaling interrupts, depending on the I/O protocol.

Many I/O devices use a signal that is activated when the device needs to interrupt for servicing. In this case a way is needed to transport this signal from the device, across the SAN to the processing node. In the preferred embodiment, the interrupt is packetized in a way that is similar to the data packetization described above, with the TCA generating a packet when the interrupt goes from the inactive to the active state, which then gets sent across the SAN to the HCA. The HCA interprets the pack and interrupts the processor in a way that is specific to the processor implementation. When the device driver software has processed the interrupt, it then needs a way to signal both the HCA and the TCA that the operation is complete, so that the controllers at both ends can reset themselves for the next interrupt. This is generally accomplished by a special end of interrupt (EOI) instruction in the software that is interpreted by the HCA to determine that the operation is complete. The HCA then is required to packetize that EOI and send it to the TCA that is handling that interrupt. The process of sending this interrupt across the SAN is similar to the data case. That is, the foreign protocol is encapsulated into the SAN protocol data packet with the appropriate headers and trailers to ensure that the data packet is delivered across the SAN to the appropriate TCA. This foreign protocol for the interrupts may be, for example, similar to what is described in Patent 5,701,495 entitled "Scalable System

22 01 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048

30 The description of the present invention turns now
to a description of PCI transactions performed over

Docket No. AUS9-2000-0546-US1

Infiniband. **Figures 8, 9, 10, and 11** together illustrate how PCI transactions are encapsulated into packets and then decoded back into PCI transactions thus allowing PCI transactions to PCI Input/Output adapters to be performed over a packet switched network, such as, for example, one utilizing InfiniBand protocols.

With reference now to **Figure 8**, a flowchart illustrating an exemplary method of performing a store operation issued by a processor to a PCI I/O adapter over InfiniBand architecture is depicted in accordance with a preferred embodiment of the present invention. When a processor, such as, for example, CPU **126** in **Figure 1**, issues a store command to a PCI I/O adapter (IOA) (step **802**), the HCA, such as, for example, HCA **118** in **Figure 1**, sees the store command and compares the address within the store command to a range of addresses on the PCI buses known to the HCA (step **804**) to determine if the address is within the range (step **806**). If the address is not within the range of addresses for the PCI buses known to the HCA, then the HCA ignores the store operation (step **818**) and the process ends since the address does not correspond to any of the PCI IOAs within the system known to the HCA.

If the address is within the range of addresses corresponding to one of the PCI IOAs within the system, then the HCA places the address and data from the store instruction along with the store command into the data payload, such as, for example, packet payload **512** in **Figure 5**, of an Infiniband (IB) data packet, such as, data packet **500** in **Figure 5** (step **808**). Thus, the PCI transaction is encapsulated into packets on the SAN.

Docket No. AUS9-2000-0546-US1

Based on the address decoded by the HCA when the store operation was observed by the HCA, the HCA determines which TCA, such as TCA 186 in **Figure 1**, contains the PCI address range of the address designated by the store operation and puts the LID address of the TCA into the local routing header of the packet (e.g., header 516 of IB packet 500), creates the CRC and other components of the data packet, and places the data packet onto the IB fabric (step 810).

Once the data packet has been placed onto the IB fabric, the data packet is routed to the correct TCA (step 812) which recognizes and accepts the data packet (step 814). The TCA then decodes the data packet payload and creates a write operation on the PCI bus with the given data and address for the PCI IOA specified by the processor (step 816).

With reference now to **Figure 9**, a flowchart illustrating an exemplary method of performing a load operation from a processor to a PCI IOA is depicted in accordance with the present invention. When a processor, such as, for example, CPU 126 in **Figure 1**, issues a load command to a PCI I/O adapter (IOA) (step 902), the HCA, such as, for example, HCA 118 in **Figure 1**, sees the load command and compares the address in the load command to the range of addresses of the PCI buses known to the HCA (step 904) and determines whether the address in the load command is within the range (step 906). If the address in the load command is not within the range of addresses of PCI buses known to the HCA, then the load command is ignored (step 926) and the process ends.

Docket No. AUS9-2000-0546-US1

00773489-120700
002021-884260

If the address within the load command is within the range of addresses for the PCI buses known to the HCA, then the HCA places the address from the load instruction along with the load command into the data packet payload of an IB packet, such as, for example, data packet payload **512** of IB packet **500** in **Figure 5** (step **908**). Based on the address in the load command, the HCA determines which TCA, such as, for example, TCA **186** in **Figure 1**, contains the appropriate PCI address range and then places the LID address of the TCA into the local routing header of the packet, such as, for example, header **516** of IB packet **500** in **Figure 5**, creates the CRC as well as other components of the data packet, and places the data packet onto the IB fabric (step **910**). The data packet is then rerouted to the correct TCA where the TCA recognizes and accepts the data packet (step **912**).

Once the TCA has accepted the data packet, the TCA decodes the packet data payload and creates a read operation on the PCI bus with the given data and address of the load operation (step **914**). The TCA waits for the data from the PCI-X IOA (step **916**) or alternatively, if the IOA is a PCI IOA rather than a PCI-X IOA, the HCA retries the IOA until it receives the requested data from the IOA. Once the requested data has been received by the TCA, the TCA places the requested data and the Load Reply command into the data payload of an IB packet (step **918**). The TCA then places the LID of the requesting HCA, which is remembered from the initial Load request packet, into the return IB packet header, creates the CRC and other components of the data packet, and then places the

Docket No. AUS9-2000-0546-US1

data packet onto the IB fabric (step 920). The data packet is then routed to the correct HCA which recognizes and accepts the packet (step 922). The HCA then decodes the packet payload and creates and sends the Load Reply
5 to the processor (step 924).

With reference now to **Figure 10**, a flowchart illustrating an exemplary method of performing a direct memory access read (DMA) operation from a PCI I/O adapter to system memory across an Infiniband system is depicted
10 in accordance with a preferred embodiment of the present invention. When a PCI or PCI-X IOA issues a direct memory access write command to the PCI bus (step 1002), the TCA sees the write and places the address and data from the write along with the write command into the data
15 payload of an IB packet (step 1004). The TCA then places the LID address of the HCA which will handle the access to the system memory into the local routing header of the packet, such as, for example, header 516 of IB packet 500 in **Figure 5**, creates the CRC and other components of the
20 data packet, and places the data packet on the IB fabric (step 1006). The packet is then routed to the correct HCA where the HCA recognizes and accepts the data packet (step 1008). The HCA decodes the packet and determines that it is a PCI operation rather than a queue pair
25 operation (step 1010). Once the HCA determines that the data packet is a PCI operation, the HCA further decodes the packet payload and creates a write operation to the system memory with the given data and address (step 1012), thus completing the DMA write operation from a PCI
30 I/O adapter to the system memory.

With reference now to **Figure 11**, a flowchart

Docket No. AUS9-2000-0546-US1

illustrating a direct memory access read operation from a PCI I/O adapter to system memory across an InfiniBand connection is depicted in accordance with a preferred embodiment of the present invention. When the PCI I/O adapter issues a DMA read operation to the PCI bus (step 1102), the TCA sees the read and places the address from the read along with the read command into the data payload of an IB packet (step 1104). The TCA then places the LID address of the HCA which will handle the access to the system memory into the local routing header of the packet, such as, for example, header 516 of IB packet 500 in Figure 5, creates the CRC as well as other components of the data packet, and places the data packet onto the IB fabric (step 1106). The data packet is then routed to the correct HCA which then recognizes and accepts the data packet (step 1108).

Once the HCA has accepted the data packet, the HCA decodes the packet and determines that the data packet is a embedded PCI transaction rather than a Queue Pair operation (step 1110). The HCA then further decodes the packet payload and creates and sends a PCI read operation to the system memory with the given data and address as extracted from the data payload (step 1112). The HCA waits for the data to be returned from the system memory (step 1114) and when received, placed the requested data and the Read Reply command into a data payload of an IB packet (step 1116). The HCA then places the LID of the requesting TCA into the return IB local routing header of the packet, such as, for example, header 516 of IB packet 500 in Figure 5, creates the CRC and other components of the data packet, and places the data packet onto the IB

[illegible]

5

10

20

25

30

Docket No. AUS9-2000-0546-US1

5 applications.

to other types of bus protocols as well, such as, for example, ISA protocols. Therefore, the present invention is not limited to application to PCI and PCI-X adapters, but may be applied to other types of I/O adapters as well. In addition, this invention is not limited to the application only to I/O, but may be applied to other operations and configurations such as Cache Coherent Non-Uniform Memory Access (NUMA or CCNUMA) and Scalable Coherent Memory Access (SCOMA) applications.

invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

THE

The description of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.